<2020. 01. 20>

1. 주석(comment)
    1) 소스에 설명을 추가하는 것
    2) 프로그램 수행에 영향이 없다.
    3) 한 줄 주석 : # 뒤부터 주석 처리
    4) 여러 줄 주석 : ''' ''' 또는 """ """(작은 따옴표 3개 또는 큰 따옴표 3개)

2. print()함수
    1) 화면으로 데이터 출력
        > 함수 : 특정 행위를 하기 위해 미리 만들어 놓은 기능
        > 화면 : IDLE이나 콘솔(cmd)
    2) 대화형 인터프리터에서는 사용 안 함
    3) 세미콜론(;)으로 여러 줄의 코드를 합칠 수 있다.
        > 한 줄에는 하나의 '명령'을 사용
        > print("A","B");print("C","D") = print("A","B")
                                          print("C","D")

3. 문자열(string)
    1) 문자의 나열
    2) 문자열을 만드려면 "ABC", 또는 'abc'처럼 따옴표로 묶는다.

4. 구분기호(기본값은 공백 sep=' ')
    ex) print(1,2,3,sep='하하') -> 1하하2하하3하하

5. 마지막 기호(기본값은 줄바꿈 end='\n') (n=new line, 새로운 줄)
    ex) print("안녕","바보","까까",sep='뿅',end='ㅋㅋㅋ')
    ex) print("하세요",end='호호호') -> 안녕뿅바보뿅까까ㅋㅋㅋ하세요호호호

6. 숫자 + 숫자, 문자 + 문자는 가능 / 숫자 + 문자 불가능
    ex) print(1+2) -> 3
    ex) print("1"+"2") -> 12

7. + 기호는 '연산' -> 하나의 값을 만든다.
   , 기호는 '나열' -> 여러 값을 나열하여 출력

8. 변수(Variable)
    1) 값을 저장하는 공간

2) 파이썬에서는 사용하는 변수는 값을 '저장'하는 개념이 아니라 값을 '가르킨다'
3) 프로그래밍 언어에서 = (equal,등호) 기호는 '같다'가 아닌 '대입'
> 우측 값을 좌측에 대입
4) a="1", 1을 저장하는 게 아니라, 어딘가에 존재하는 1을 a가 가르키고 있다
5) 파이썬에서는 변수가 어떤 형태의 값이든 가리킬 수 있다.
> 값의 형태를 명시하지 않고 사용
6) 하나씩 대입, a=1
7) 한 번에 대입, 순서대로 대입, 짝이 맞아야 함, a=b=c=7
8) 변수끼리 값 교체, a=1, b=2 -> a,b=b,a
9) 변수가 값을 가르킨다
ex) a=10, b=10
print(id(10)) = print(id(a)) = print(id(b))
10) import sys > sys라는 '모듈'을 추가
print("처음 2020 가르키는 개수 :",sys.getrefcount(2020))
> sys 모듈에 존재하는 함수를 사용
> get: 구하겠다
> ref: reference(참조하는)
> count: 개수
= 즉, 숫자 2020을 가르키고 있는 개수
11) 변수명 규칙
① 한글 사용 가능 -> 그래도 영어로 한다.
② 특수문자(기호)는 _만 사용
③ 숫자 사용 가능, 단 첫 글자는 안 됨
(test123=1 -> 가능, 123test=1 -> 불가능)
④ 대소문자 구분
⑤ 예약어 사용 안 됨
⑥ 중요함
> 변수명 작성 시 의미 부여
> 변수만 봐도 어떤 값을 사용하는 지 알 수 있도록
⑦ 숫자 num
⑧ 문자 str
12) del() 변수를 지우는 명령어

9. 예약어 목록 확인
import keyword -> keyword라는 모듈을 사용하겠다.
> print(keyword.kwlist) : 파이썬에서 쓰이는 예약어 목록 볼 수 있음

<2020. 01. 21>

<자료형 - 어떠한 값(자료)의 형태>

1. 숫자형(Number)

1) 정수 : -1234, 0, 1234
2) 실수(소수) : 1.1, 3.14, -123.331
3) 2진수(binary) : 0~1 / 0b10, 0B10
4) 8진수(octal) : 0~7 / 0o10, 0O10
5) 16진수(hex) : 0~9, a~f / 0x10, 0X10
6) 사칙연산 : + - * /
7) 나머지 연산 : %
    > 두 수를 나누고 나머지 값만 사용
8) 몫연산 : //
9) 제곱연산 : **

- 연산자 : 연산(계산)을 수행하는 '기호'
- 피연산자 : 연산자의 작업 대상
- 연산을 수행한다 -> 피연산자를 이용해서 하나의 값을 만듦
    * 연산을 수행할 때 소괄호로 묶는다.(필수는 아니나 좋은 습관)

ex) num1=10 / num2=3
    print("num1 - num2 =",(num-1num2)) -> num1 - num2 = 7

2. 문자형(string)

1) 따옴표로 둘러싸이면 무조건 문자열
2) 문자열을 만드는 방법
    ① 큰 따옴표 1개 ex) print("happay day")
    ② 작은 따옴표 1개 ex) print('happy day')
    ③ 큰 따옴표 3개 ex) print('''happy day''')
    ④ 작은 따옴표 3개 ex) print("""happy day""")
3) 이스케이프 문자
    > 문자열 안에서 특수한 기능을 가지는 문자
    > 역슬래시(\)로 시작한다
        \n : 개행(줄바꿈) new Line
        \t : tab 키를 누른 만큼 들여쓰기
        \\ : \ 출력
        \" : " 출력
        \' : ' 출력

* 문자열에 같은 따옴표 중복 안 됨

ex) print("김철수 : "파이썬 재밌다"") -> error!

print('김철수 : "파이썬 재밌다"')

print("김철수 : '파이썬 재밌다'")

print("김철수 : \"파이썬 재밌다\"")

* 여러 줄의 문자열 다루기 -> 따옴표 3개짜리 사용

ex) print("""안녕하세요.

파이썬입니다.""")

= print("안녕하세요.\n파이썬입니다.")

4) 문자열 연산하기

> + : 연결 / * : 반복

ex) print("안녕"+"하세요.") -> 안녕하세요.

str1="안녕" / str2="하세요"

print(str1+str2) -> 안녕하세요.

ex) print("안녕"*3) -> 안녕안녕안녕

3. 문자열 인덱싱

1) 인덱싱(indexing) : index 색인, 무언가를 가리킨다,

2) 문자열에서 특정 글자를 뽑아내어 사용하는 것

> 특정 글자를 찾을 때 '순서'를 사용 -> 인덱스

> 순서는 0부터 시작

>> 인덱스라는 표현(용어)이 들어가면 무조건 0부터

> 음수는 뒤에서부터 순서를 센다(-1부터 시작)

> index 범위를 초과하면 error!

ex) my_str="I Love You" (띄어쓰기 포함 10글자)

print(my_str[6]) -> e

print(my_str[-1]) -> u

print(my_str[2],my_str[5],my_str[9]) -> L e u

4. 문자열 슬라이싱

1) 슬라이싱(slicing) : 조각낸다.

2) 콜론(:)으로 범위 지정 ex) a[0:3] -> 0에서 3까지

ex) my_str="I Love You" (띄어쓰기 포함 10글자)

print(my_str[2:6]) -> Love

3) [시작 인덱스:끝 인덱스] -> 끝 인덱스는 포함 안 됨
4) [시작 인덱스:] -> '시작 인덱스'부터 '끝'까지
5) [:끝 인덱스] -> '처음'부터 '끝 인덱스'까지(끝 인덱스는 포함 안 됨)
6) [:] -> 시작부터 끝 = 전체 = a
7) 문자 범위를 초과해도 오류 아님
    ex) 10글자 짜리 인덱스인데 [0:100] -> 오류 아님
8) 음수로 문자 범위를 초과해도 오류 아님, 대신 아무것도 표시되지 않음.
    ex) [:-100]
9) 중요! 문자열 변경 불가능 -> 새로 만들어야 함
    ex) my_str="I Love You" (띄어쓰기 포함 10글자)
        new_str="i"+my_str[1:6]+my_str[-4:]
        print(new_str) -> i Love You
10) 하나의 자료(문자열, 리스트 등)가 여러 개의 값으로 이루어진 경우 사용
11) 문자열 = 문자 하나하나가 나열된 자료

<2020. 01. 22>

1. 문자열 기본 포매팅
    1) 문자열 안에 '값'을 '삽입'하는 방법
    2) 포맷코드(서식 문자)
        ① %s : 문자열(string)
            ex) print("문자열 : %s"%"나는 문자열") -> 문자열 : 나는 문자열
            > 모든 것을 글자 취급함
            > 숫자를 문자 형태로 삽입 가능
        ② %c : 문자 1개
        ③ %d : 정수(소수점 이하 소멸, 문자열 삽입 불가)
            ex) print("정수 : %d입니다." %10)
            ex) my_str = "정수 : %d" %20
                print(my_str) -> 정수 : 20
        ④ %f : 실수
            ex) print("실수 : %f입니다."%10.123) -> 실수 : 10.123000입니다.
            > 정수 입력하면 없던 소수점 6자리 생김
            > 소수점 7번째 자리에서 반올림
            > 문자열 삽입 불가
            > 소수점 자리를 N개 표시하고 싶으면 %.Nf
                ex) print("세 자리 : %.3f"%4.46788) -> 세 자리 : 4.468
        ⑤ %% : % 하나 삽입

    3) 여러 개의 값을 넣으려면 % 뒤의 값들을 순서대로 소괄호로 묶어준다.
        ex) print("안녕하세요. %s입니다. %d살이에요."%("최은실",29))
                -> 안녕하세요. 최은실입니다. 29살이에요.

    4) 포매팅을 사용할 때에는 %% 중복 불가!
        ex) print("%d% 과일주스"%100) -> error!
            print("%d%% 과일주스"%100) -> OK (100% 과일주스)

    ex) 변수를 사용하는 경우
        year=2020
        print("올해는 %d년입니다."%year) -> 올해는 2020년입니다.

2. 포맷 코드를 활용한 정렬과 공백
    1) 공백 삽입할 때 %와 s 사이에 숫자 입력(오른쪽에 공백 삽입 원할 때는 음수)

> 공백 개수는 숫자 - 글자 수(N칸 확보 후 값을 대입)

ex) print("%10s"%"안녕") -> [안녕] (공백 8개)

ex) print("%-10s"%"안녕") -> [안녕] (공백 8개)

3. 문자열 포매팅 함수

1) 포맷 코드 대신 {} 중괄호를 사용한다.

2) 하나의 문자열을 만드는 과정

3) .format() 함수는 '문자열'로 사용할 수 있는 '문자열의 함수'

4) 기본은 순서대로

ex) print("제 이름은 {}이고 {}살입니다.".format("최은실",29))

-> 제 이름은 최은실이고 29살입니다.

5) {} 안에 인덱스 사용할 때는 항상 0부터 시작

ex) print("제 이름은 {1}이고 {0}살입니다.".format("최은실",29))

6) 키워드 사용

ex) name = "최은실" / age = 29

print("제 이름은 {}이고 {}살입니다.".format(name,age))

ex) print("제 이름은 {name2}이고 {age2}살입니다.“

.format(name2="최은실",age2=29))

ex) print("제 이름은 {name2}이고 {age2}살입니다.“

.format(name2=name,age2=age))

ex) print("제 이름은 {name2}이고 {age2}살입니다.“

.format(age2=age,name2=name))

> 키워드와 인덱스 혼용 시, 키워드는 맨 뒤에 위치

ex) print("제 {}은 {name2}이고 {}살입니다.“

.format("이름",29,name2=name))

ex) print("제 {}은 {name2}이고 {}살입니다.“

.format("이름",name2=name,29)) -> error!

7) 소수점 표현

ex) print("소수 : {}".format(10.123)) -> 소수 : 10.123

> 문자열 취급하므로 여섯짜리까지 나오지 않음

ex) print("소수 : {}".format(10.23423553653764524332342))

> 그래도 너무 길면 14자리까지만 나옴

> 소수점 N자리를 표현하고 싶을 경우 "{:.3f}" 또는 "{0:.3f}"

ex) print("{:.3f}".format(10.1232545345)) -> 10.123
또는 print("{0:.3f}".format(10.1232545345))

8) format() 함수 사용 시 중괄호 {}에 특수한 기능을 추가하는 기호 -> 콜론
> 콜론을 사용할 때에는 인덱스 뒤에 위치(인덱스 생략 가능)

9) 정렬
> 좌측 정렬(기본, <)
ex) print("[{:10}][{:10}]".format("안녕", "춥다"))
-> [안녕][춥다]
print("[{:<10}][{:<10}]".format("안녕", "춥다"))
> 사실은 "<"가 생략되어 있음(기본이라서)
> 우측 정렬(>)
ex) print("[{:>10}][{:>10}]".format("안녕", "춥다"))
-> [안녕][춥다]
> 가운데 정렬(^)
ex) print("[{:^10}][{:^10}]".format("안녕", "춥다"))
-> [안녕][춥다]
> 정렬 후 빈 공간에 값 채우기(기호 앞에 값 넣기)
ex) print("[{:a^10}]".format("춥다"))
-> [aaaa춥다aaaa]

4. 문자열 관련 함수
1) "".format()처럼 문자열을 이용해서 사용할 수 있는 유용한 함수
2) XX 관련 함수 : xx.함수() 문법 규칙임
3) upper() : 문자열의 영문을 모두 대문자로 바꿈
ex) str1="hello"
print(str1.upper()) -> HELLO(str1 자체가 바뀌는 건 아님)

4) lower() : 문자열의 영문을 모두 소문자로 바꿈
ex) str2="HELLO"
print(str2.lower()) -> hello(str2 자체가 바뀌는 건 아님)

5) title() : 문자열을 제목처럼 각 영단어의 앞 글자만 대문자로 바꿈
ex) str3="i love you"
print(str3.title()) -> I Love You

6) strip() : 문자열 좌우측에 존재하는 공백 제거(문자 사이의 공백은 제거 안 됨)

ex) str4=" I Love You "
print(str4.strip()) -> I Love You

> 좌측 공백 제거(lstrip)

ex) str5=" 이 럴 수 가 "
print("좌측 공백 제거 :",str5.lstrip()) -> 이 럴 수 가

> 우측 공백 제거(rstrip)

ex) print("우측 공백 제거 :",str5.rstrip()) -> 이 럴 수 가

7) join() : 특정 문자열을 대상 문자열에 삽입

ex) print("A".join("BBB")) -> BABAB

ex) a = ","
print(a.join("문자열 삽입 join()")) -> 문,자,열, ,삽,입, ,j,o,i,n,(,)
print(",".join("JOIN")) -> J,O,I,N

8) count("A") 문자열에서 "A"의 개수를 반환(함수의 결과 값이 A의 개수)

ex) str5="python python python"
print("str5에서 p의 개수 :",str5.count("p"))
-> str5에서 p의 개수 : 3 (없으면 0)
print("str5에서 p의 개수 :"+str5.count("p"))
-> error! 문자 + 숫자 안 됨!

5. 복합 대입 연산자

1) +=, -=, *= 대입 연산자(=)와 다른 연산 기호가 합쳐진 형태
2) 나 자기자신의 값을 이용해서 연산 후, 나한테 다시 대입
3) 주의사항 : 복합 대입 연산자 사용 시 사용할 변수는 만들어져 있어야 한다.

ex) a+=b = a=a+b
a-=b = a=a-b
a*=b = a=a*b
a**=b = a=a^b

<2020. 01. 23>

1. 문자열 관련 함수

1) replace("A", "B") : 문자열에서 모든 "A"를 찾아서 "B"로 변경

ex) str3 = "python python python"
str4 = str3.replace("py", "Py")
print(str4) -> Python Python Python

2) split("A") : 문자열을 기준 문자 "A"로 나눈다.

> split() 안에 아무 값도 넣지 않으면 기본이 공백, 개행 등으로 나눈다.
> 나오면 결과는 리스트 자료형

ex) str5 = "문자열 나누기(split)"
print(str5.split()) -> ["문자열", "나누기(split)"]
print(str5.split("(")) -> ["문자열 나누기", "split)"]

3) index("A") : 문자열에서 "A"를 찾고 그 위치를 반환(구분은 + 말고 ,로 할 것)

> 없는 문자라서 찾지 못하면 error!

ex) str6 = "문자열 위치 찾기(index)"
print("str6에서 '열'의 위치 :",str6.index("열")) -> 2
print("str6에서 'index'의 위치 :",str6.index("index")) -> 11
print("문자열문자열".index("문", 2)) -> 3

4) find("A") : 문자열에서 "A"를 찾고 그 위치를 반환

ex) print("abcdefg".find("a")) -> 0

> 없는 문자라도 오류 나지 않고 -1로 표시됨

ex) print("abcdefg".find("z")) -> -1

> 찾고자 하는 문자열이 여러 개인 경우 처음 찾은 위치만 표시됨

ex) print("문자열문자열".index("문")) -> 0

> rifnd("A") - reverse, 뒤에서부터 셈(1부터 시작)

ex) print("문자열문자열".rindex("문")) -> 3

2. 리스트(list)

1) 데이터(값)들의 목록
2) 편리하다, 리스트를 사용하겠다. = 관련 있는 자료끼리 묶겠다.
3) 리스트 종류

- 요소가 없는 빈 리스트 ex) a = []
- 요소가 정수 ex) b =[1, 2, 3]

- 문자열 ex) c = ["A","B", "C"]
- 혼합 ex) d = [1, 2, "A", "B"]
- 혼합 + 리스트 안에 또 리스트 ex) e = [1, "A", [2, "B"]]
↳ 혼합은 문법적으로는 문제 없으나 바람직한 형태가 아님

4) type : 자료의 종류를 확인
ex) print("변수 a의 type :", type(a)) -> 변수 a의 type : <class 'list'>
ex) print("1의 type :","tpye(1)") -> 1의 type : <class 'int'>

5) 리스트 인덱싱, 슬라이싱
> 문자열 : 하나 하나 문자들이 순서대로 나열된 형태
> 리스트 : 하나 하나 요소들이 순서대로 나열된 형태
> 순서가 있다 - 인덱싱, 슬라이싱이 가능
ex) my_list = ["한수창", "홍길동", "이몽룡"]
print("첫 번째 사람은", my_list[0]) -> 첫 번째 사람은 한수창
↳ 첫 번째 요소가 '문자열'이기 때문에 인덱싱 결과도 '문자열'
ex) num_list = [1, 2, 3]
print("첫 번째 요소 :", num_list[0]) -> 첫 번째 요소 : 1
↳ 요소가 '정수'이기 때문에, 인덱싱의 결과도 '정수'

> 인덱싱 비교
- 문자열 : 모든 요소가 문자이기 때문에 인덱싱하면 다 '문자'
- 리스트 : 각 요소의 형태에 따라 인덱싱 결과가 다르다.

> 이중 리스트 인덱싱(리스트 안에 리스트)
ex) my_list = ["한수창", "홍길동", ["임꺽정", "이몽룡"]]
print(my_list[2]) -> ['임꺽정', '이몽룡']
print(my_list[2][1]) -> 이몽룡

> 문자열과 숫자 구분
ex) a = [2, "2"]
print(a[0]*2) -> 4
print(a[1]*2) -> 22

> 슬라이싱
ex) a = [2, "2"]
print(a[0:2]) -> [2, '2']
print(a[0:1]) -> [2] (요소가 하나여도 슬라이싱 결과는

무조건 리스트)

print(a[0]) -> 2

6) 리스트 연산하기

> 리스트 덧셈 = 연결, 리스트 곱셈 = 반복

ex) a = [1, 2, 3]
b = [4, 5, 6]
print(a+b) -> [1, 2, 3, 4, 5, 6]
print(a*2) -> [1, 2, 3, 1, 2, 3]

> 하나의 새로운 리스트 생성

ex) c = a + b
print(c) -> [1, 2, 3, 4, 5, 6]

7) 리스트 수정하기

ex) a = [1, 2, 3, 4, 5, 6]
a[2] = -1
print(a) -> [1, 2, -1, 4, 5, 6]

① 연속된 범위의 값을 수정할 때는 항상 '리스트'로

ex) a[0 : 2] = 0 -> error!
a[0 : 2] = [0]
print(a) -> [0, -1, 4, 5, 6]

ex) a[0 : 2] = [6, 7, 8]
print (a) -> [6, 7, 8, 4, 5, 6]

② 인덱싱

ex) a[0] = [1, 2]
print (a) -> [[1, 2], 7, 8, 4, 5, 6]

③ 삭제

ex) a[0] =[] (삭제가 아니라 빈 리스트 대입)
print (a) -> [[], 7, 8, 4, 5, 6]

- del : 해당 요소 제거

ex) del(a[0])
print (a) -> [7, 8, 4, 5, 6]

ex) del(a[0:1])
print (a) -> [8, 4, 5, 6]

ex) del(a)
print (a) -> error! (a 자체가 지워져서)

- 슬라이싱으로 제거 : 빈 리스트 대입

ex) a[0:2] = []

print (a) -> [5, 6]

8) 리스트 관련 함수(리스트.함수())

① append(value) : 리스트 가장 뒤에 value를 추가
: 새로운 리스트를 만드는 것이 아니라 기존 리스트를 수정
: 하나의 요소만 추가 가능

ex) a = [1, 2, 3]
a.append(4)
print(a) -> [1, 2, 3, 4]

ex) a.append([5, 6])
print(a) -> [1, 2, 3, 4, [5, 6]]

② sort() : 리스트 정렬(숫자, 알파벳 등)

ex) a = [9, 2, 8, 1]
a.sort() (기본 오름차순)
print(a) -> [1, 2, 8, 9]

ex) a.sort(reverse = true) (정렬 후 결과를 뒤집는다, 내림차순)
print(a) -> [9, 8, 2, 1]

> a.sort()와 sorted(a)
- a.sort()는 a 자체가 정렬(a가 주체)
- sorted(a)는 a를 정렬한 새로운 리스트 생성(a는 도구)
ex) a = [3, 1, 2]
b = sorted(a)
print(a) -> [1, 2, 3]
print(b) -> [3, 1, 2]

③ reverse() : 리스트 뒤집기(현재 요소를 그대로 뒤집는다)

ex) a = [9, 2, 8, 1]
a. reverse()
print(a) -> [1, 8, 2, 9]

④ index() : 리스트에서 요소를 찾고 그 위치 반환

ex) a = [1, 2, 3]
prinit(a.index(2)) -> 1

⑤ insert(index,value) : 지정한 위치에 값 삽입

ex) a.insert(1, "뽕")
print(a) -> [1, '뽕', 2, 3]

⑥ remove() : 리스트에서 처음 찾은 값 제거
: 없는 값 입력하면 error!

ex) a = [1, 2, 3, 1]

a.remove(1)
print(a) -> [2, 3, 1]

⑦ count() : 리스트에 존재하는 요소의 개수 반환

ex) a = [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 1, 2, 3, 1, 2, 3, 1, 2, 3]
print(a.count(2)) -> 6

⑧ pop(index) : 매우 중요!

: 리스트에서 (index)번째 값을 "뽑아낸다."
: 뽑아낸 값을 '반환'해준다.
> 우리가 사용할 수 있다. (변수에 대입 or 어딘가에 사용)
: 리스트에서 해당 값을 제거(뽑아내는 개념)

: index를 비워두면 기본 맨 뒤

ex) a = [1, 2, 3, 1]
print(a.pop(1)) -> 2
print(a) -> [1, 3, 1]

ex) print(a.pop()) -> 1
print(a) -> [1, 3]

⑨ len() : 요소의 개수를 구하는 함수

: 값이 여러 개 존재하는 자료형만 사용 가능

ex) a = [1, 2, 3, 4] 리스트 = (어떤 값들이 여러 개 존재)
b = "1234" 문자열 = (문자들이 여러 개 존재)
c = 1234 정수 = 단 하나의 숫자(값)
d = 1234, 567 정수가 여러 개
print(len(a)) -> 4
print(len(b)) -> 4
print(len(c)) -> error!
print(len(d)) -> 2

⑩ copy() : 모든 값들을 '복제'하여 새로운 리스트 생성

: 복제랑 대입 잘 구분할 것!

ex) a = [1, 2, 3, 4]
b = a.copy()
c = a
print(a) -> [1, 2, 3, 4]
print(b) -> [1, 2, 3, 4]
print(c) -> [1, 2, 3, 4]

b[0] = -10
c[0] = -20
print(a) -> [-20, 2, 3, 4]

print(b) -> [-10, 2, 3, 4]
print(c) -> [-20, 2, 3, 4]

⑪ clear() : 리스트의 모든 요소 제거

: 요소만 제거되고 리스트는 삭제되지 않음

ex) a.clear()
print(a) -> []

⑫ join() : 리스트의 요소들이 "문자열"로만 이루어진 경우,
하나의 문자열 생성 가능

ex) my_list = ["대", "한", "민", "국"]
my_str = "".join(my_list)
print(my_str) = 대한민국

<2020. 01. 28>

1. 튜플(Tuple) 자료형
    1) 리스트와 비슷
        - 생성법
            > 리스트 : []
            > 튜플 : ()
        - 튜플은 한 번 만들면 변경 불가능(문자열처럼)
    2) 프로그램이 수행되는 동안 요소가 변경되지 않게 하고 싶다면
                                                리스트 대신 튜플을 사용한다.
    3) 일반적으로 '리스트' 사용
    4) 튜플 형태
        ex) a = ()   -> 빈 튜플(쓸모 없음)
        ex) b = (1,)   -> 요소가 한 개 일 땐 뒤에 콤마를 붙인다.
        ex) c = (1, 2, 3, "A", "B", "C")
        ex) d = 1, 2, 3   -> () 생략해도 튜플
        ex) e = (1, "A", (3, "B"))
        ex) a, b, c = 1, 2, 3   ->이건 변수
    5) 튜플은 변경이 불가함
        ex) a = 1, 2, 3, 4
            print(a)   -> (1, 2, 3, 4)
            print(a[0])   -> 1
            print(a[0:2])   -> (1, 2) (슬라이싱 가능)
            a[0] = -1   -> error!
            del(a[0])   -> error!
        ex) a = 1, 2
            b = 3, 4
            c = a + b
            print(c*2)   -> (1, 2, 3, 4, 1, 2, 3, 4)

2. 딕셔너리(Dictionary) 자료형
    1) 형태 : {key1 : value, key2 : value2...}
        > key와 value는 한 쌍
    2) 딕셔너리는 '순서'가 없고 'key'를 가지고 인덱싱
    3) 주의사항
        - key 값은 중복되면 안 된다.
        - key 값으로 리스트, 딕셔너리 사용 불가
        - key : 변하지 않는 성질의 자료(값)

- value : 아무거나 상관없음

ex) my_dict = {"과일":"사과", "가격":1000, (1,2):("원","투"),"개수":[3,4]}

print(my_dict["과일"]) -> 사과

print(my_dict["개수"][0]) -> 3

print(my_dict[(1,2)][1]) -> 투

4) 딕셔너리 추가, 삭제

- my_dict[새로운 key] = 새로운 value

ex) my_dict = {"과일" : "사과"}

my_dict["채소"] = "파"

print(my_dict) -> {'과일' : '사과', '채소' : '파'}

- del(my_dict[key])

ex) del(my_dict["과일"])

print(my_dict) -> {'채소' : '파'}

5)문자열 포매팅 함수에서 키워드 사용

ex) "{name}".format(name="한수창")

3. 집합(Set) 자료형

1) 수학에서의 집합을 의미

> 교집합, 합집합, 차집합 등을 구할 수 있다.

2) 중복된 값을 허용하지 않는다.

> 중복 제거 용도로 사용

ex) my_set = {1, 2, 3, 1, 1, 2, 2, 3, 3,}

print(my_set) -> {1, 2, 3}

3) 순서가 없다. (인덱싱 불가능)

4) set() : 집합으로 변환해주는 함수

4) 비슷한 성질의 자료끼리 변환

ex) my_str = "Hello"

my_set = set(my_str)

print(my_set) -> {'H', 'e', 'l', 'o'}

5) 집합에서 특정 값을 인덱싱하고 싶을 경우 리스트나 튜플로 변환해서 인덱싱

ex) my_list = list(my_set)

print(my_list[0]) -> e

4. bool 자료형

1) 참(True), 거짓(False)을 표현하는 자료형

2) 거짓인 경우

- 요소가 없다. (문자열, 리스트 등)

- 숫자가 0이다. (0만 아니면 다 참)

- none : 값이 없다는 걸 의미하는 '자료형/값'

3) True/False 2개의 값만 존재

4) '조건식'을 다룰 때 나오는 개념

ex) print(bool(0)) -> False
print(bool(1)) -> True
print(bool(-1)) -> True
print(bool([])) -> False

5. 자료형 형태 바꾸기

1) int() : 정수로 변환(숫자끼리만 가능)
2) float() : 실수로 변환(숫자끼리만 가능)
3) tuple() : 튜플로 변환
4) dict() : 딕셔너리로 변환
5) list() : 리스트로 변환
6) set() : 집합으로 변환

<2020. 01. 29>

1. input()

1) 입력 대기 상태 -> 입력 후 엔터

> input() 함수에 의해 입력된 내용이 '문자열'로 반환, 변수에 대입 가능

2) input("입력 받기 전 출력할 문자열")

3) 입력과 변환을 한 번에

```
ex) input_num1 = int(input("숫자1 입력 : "))
    input_num2 = int(input("숫자2 입력 : "))
    print("두 수의 합 : ",(input_num1 + input_num2))
```

4) map()과 함께 쓰기

- 좌측에 나열된 변수의 개수와 map의 요소 개수가 일치해야 함

```
ex) a, b = map(int,["1", "2"])     -> a, b=int("1"), int("2") -> a, b = 1, 2
ex) num1, num2 = map(int, input("두 수 입력 :").split)
ex) num1, num2, num3 = map(int, input("세 수 입력 :").split)
```

```
ex) 키가 얼마인가요? 165
    몸무게는 얼마인가요? 50.0
    키 + 몸무게 =  215.0
    height = input("키가 얼마인가요? ")
    height = int(input("키가 얼마인가요? "))
    weight = float(input("몸무게는 얼마인가요? "))
    print("키 + 몸무게 =",float(height+weight))
```

```
ex) 당신의 나이는 몇 살입니까?   30
    당신은 30 년을 살았습니다.
    age = input("당신의 나이는 몇 살입니까? ")
    print("당신은",age,"년을 살았습니다.")
    print("당신은 {} 년을 살았습니다.".format(age))
```

```
ex) 정수를 입력하세요 : 10 20 30
    60
    num1, num2, num3 = map(int,input("정수를 입력하세요 : ").split())
    print(num1+num2+num3)
```

2. if문 기본 구조

1) if / elif / else 조건식 :

```
    수행문
```

2) 조건식
- 참(True)이나 거짓(False)으로 판별 가능해야 한다.
- 조건식 끝에는 콜론(:)을 붙인다.(콜론이 있으면 조건식 끝)
- 콜론(:) 뒤부터는 수행문으로 간주한다.

3) 수행문
- 반드시 '들여쓰기'를 해야한다(or 공백 4개)
- 들여쓰기만 맞으면 수행문 여러줄 작성 가능
- 들여쓰기가 끝나면 if문의 수행문이 끝난 것

4) elif 조건식 : (else if) 다른 만약에
- 그게 아니라면 만약 ~ 이 조건은?
- if문 종속(if문 없이 단독 사용 불가)
- 조건문의 시작은 무조건 if문

5) else
- 위 조건식을 모두 만족하지 않으면 무조건 이것을 수행
- 따로 조건식이 없다.
- if문 종속
- 하나만 사용 가능

```
ex) password = input("비밀번호는 무엇입니까?")
    if password == "0529" :
      print("문이 열렸습니다.")

ex) num = int(input("숫자 입력 : "))
    if num > 0 :
      print("양수")
    elif num < 0 :
      print("음수")
    else : #위 조건에 만족하지 않으면 실행됨
      print("0입니다.")
```

<2020. 01. 30>

3. if문 중첩

1) if문 수행문 앞에 또 다른 if문 작성

ex) if score >= 60 :
print("합격입니다.")
if score == 100 : -> 60점 이상인 대상 중에서 100점일 경우
print("축하합니다")
else :
print("불합격입니다.")

2) 비교연산자

- 조건식에 자주 사용되는 연산자
- 조건에 만족하면 결과값이 True, 아니면 False
- 등호(=)와 다른 연산자를 함께 사용 시 다른 연산자보다 뒤에 있어야 함
- a < b a가 b보다 작냐?
- a > b a가 b보다 크냐?
- a <= b a가 b보다 작거나 같냐?
- a >= b a가 b보다 크거나 같냐?
- a == b a가 b와 같냐?
- a != b a가 b와 같지 않냐?

3) 논리 연산자

- a or b a 또는 b 중에 하나라도 참이면 참
둘 다 거짓이어야 거짓
- a and b a와 b 둘 다 참이어야 참
둘 중 하나라도 거짓이면 거짓
- not a a가 거짓이면 참 / 참이면 거짓

4) 포함 연산자

- a in b b 안에 a가 있으면 참
- a not in b 안에 a가 없으면 참
(b에는 요소가 여러 개인 자료형의 값이 위치(리스트, 문자열 등))
- "A" in ["A", "B"] --> True
"A" in "AB" --> True
"A" in ["ABC"] --> False
> 요소 자체가 "ABC", 리스트에 "A"라는 요소는 없다.

5) 연습문제

- 주민등록번호 남/녀 판별기
주민등록번호를 010101-3456789 형태로 입력 받고,

7번째 숫자에 따라 "남자" 또는 "여자" 출력
9, 1, 3, 5, 7 : 남자
0, 2, 4, 6, 8 : 여자

[출력결과]
주민등록번호 입력 : 010101-3456789
남자

```
jumin = input("주민등록번호 입력 : ")
gender = jumin[7]
man = [1, 3, 5, 7, 9]
woman = [0, 2, 4, 6, 8]

if gender in man :
        print("남자")
elif gender in woman :
        print("여자")

        or

if (gender%2) != 0
        print("남자")
elif(gender%2) == 0
        print("여자")
```

<2020. 01. 31>

1. 반복문

1) 조건에 만족하면 수행한다.

- 단, 조건에 만족하지 않을 때까지.

2) while문

- 조건식이 참이면 수행
- if문과 기본 구조 동일
  > if문 : 조건이 참이면 수행 끝
  > while문 : 조건이 참이면 수행하고 다시 조건식을 비교
- 무한반복
  > 항상 조건이 만족하여 반복문이 무한 반복 됨
  > ctrl + c = 강제 종료
- 조건 변수
  > 조건식의 비교에 사용되는 변수
  > 조건 변수에 따라 반복 횟수가 정해짐
  > 조건변수는 미리 생성된 변수여야 함
- break : 반복문 종료
  > 무한반복 조건 걸어두고 break를 이용해 반복문 종료
  > 사용하기 위해서는 if문이 필요
- continue : 만나는 순간 조건식으로 다시 이동
  > 사용하기 위해서는 if문이 필요

ex)
```
num = 1
while num < 10 :
  if num % 2 == 0 :
        num += 1
        continue

  print("num = {}".format(num))
  num+=1
print("이 때의 num : ",num)
```

<2020. 02. 03>

1. for문

ex)

```
guest_list = [["홍길동", 19],["이몽룡", 27],["성춘향", 18], ["김철수", 29]]

num = 0
for guest in guest_list :
        name = guest[0]
        age = guest[1]
        num += 1
        print("{}번 손님 입장하실게요~".format(num))
         if age > 19 :
                print("{}님은 성인입니다. 입장하세요.".format(name))
        else :
                print("{}님은 미성년자입니다.
                        입장하셔서 우유만 드세요.".format(name))

        if age < 20 :
                continue
print("{}번째 손님인 {}님은 성인입니다.{}.세".format(num,name,age))
```

ex2)

```
for i in range(2, 10)
        print("{}"단.format(i))
        for j in range(1, 10)
        print("{} X {} = {}".format(i, j, (i*j)))
        print()
```

ex3) 뒤집기

```
for i in reversed(range(1, 10))
```

2. 랜덤

- import random 먼저 입력
- print(random.random()) -> 0.0 ~ 1.0 사이의 실수를 반환
- print(random.random()+1.0) -> 1.0 ~ 2.0 사이의 실수를 반환
- print(random.randint(1,10)) -> 1 ~ 10 사이의 정수를 반환

<2020. 02. 04>

1. 함수 (Function)

1) 특정 작업을 수행하는 일련의 문장들을 하나로 묶은 것

2) 장점

- 한 번 만들어 놓으면 언제든지 재사용 가능
- 중복된 코드 제거 가능
- 프로그램의 구조화
  > 작업 단위로 코드를 묶어서 구조화 시킨다.

3) 기본 구조

- def 함수 이름(매개 변수) :
  수행문
  수행문
  retunr 반환값

- 매개변수(parameter)
  > (필요 시) 함수가 호출될 때 값을 받는 변수
  > 개수 제한이 없고, 필요 없으면 생략도 가능
  > 우리가 함수를 호출할 때 전달하는 '값'을 인수라고 부른다.

- 반환값
  > return 뒤에 오는 값을 되돌려 줌
  > return을 사용하면 수행이 끝난다. : 마치 반복문 break와 비슷
  (뒤에 수행문이 더 있더라도 실행 안 됨)
  > return 뒤에 아무 것도 없으면 None으로 반환됨

- 유형
  > 매개변수와 반환값이 둘 다 있음
  > 매개변수와 반환값이 둘 다 없음(기능 수행만 함)
  > 매개변수만 있음
  > 반환값만 있음(만드는 함수의 목적에 따라 알아서 결정)

- 여러 값 반환하기(거짓말)
  > def calc(a, b) :
  return a+b, a-b, a*b, a/b -> 여러 개처럼 보이지만 '튜플'

- 가변인수

> 전달하는 값의 개수가 변할 수 있음

> 함수를 만드는 입장에서 변할 수 있는 값들을 처리

> 일반 매개변수, 가변인수를 혼용할 때 *args는 마지막에 위치

ex) def add(*args) :

```
    add_result = 0
    for i in args :
        add_result += i
    return add_result

print(add(1, 2,))
print(add(1, 2, 3, 4, 5))
print(add(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
```

- 매개변수에 초기값 사용

> 값을 주지 않으면 초기값이 출력됨

ex) def print_info(name, age, phone="010-xxxx-xxxx") :

```
    print("이름 :", name)
    print("나이 :", age)
    print("번호 :", phone)

    print_info("홍길동", 20, "010-1111-2222")
    print_info("임꺽정", 30, "010-2222-3333")
    print_info("성춘향",18)  ->  초기값 출력
```

- 키워드 인수

> 함수의 매개변수를 키워드로 사용

ex) def print_info(name, age, phone="010-xxxx-xxxx") :

```
    print("이름 :", name)
    print("나이 :", age)
    print("번호 :", phone)

    print_info("홍길동", 20, "010-1111-2222")
    print_info("임꺽정", 30, "010-2222-3333")
    print_info(age=19, name="성춘향", phone="112")
```

<2020. 02. 05>

1. 재귀함수

1) 함수의 수행문 안에서 나 자신을 다시 호출하는 함수
2) 수행문이 반복되기 때문에 반복문과 유사한 성격
3) 너무 많이 반복 수행하다 보면 프로그램 오류 발생
4) 함수 호출 시 'stack'이라는 구조의 메모리 사용
- Queue : First in First out(FIFO) - 입구 / 출구 따로
- Stack : First in Last out(FILO) - 출입구 하나
5) 재귀함수도 반복 호출이 끝날 수 있는 조건이 필요

ex)
```
def func(num) :
    print("func() 시작, num =", num)
    if num == 1:
        print("num1일 때 끝")
        return
    func(num - 1)
    print("func() 끝, num =", num)
func(3)
```

2. 지역변수와 전역변수

1) 지역변수 : 특정 지역에서만 사용 가능한 변수
2) 전역변수 : 전체 영역에서 사용 가능한 변수
3) glober : 지역변수를 전역변수로 쓸 때 사용

<2020. 02. 06>

1. 파일 입출력

1) 실제 파일 생성/삭제/쓰기/읽기 등의 행위

2) 파일의 이해

- 디렉토리(Directory)

> 폴더 또는 디렉토리라고 부른다.

> 폴더 안에는 파일 외 또 다른 폴더를 포함할 수 있다.

> 용량이 없다.

> 필더 안으로 들어가는 일 밖에 못함(실행이라는 개념이 없음)

- 파일

> 컴퓨터에서 정보(data)를 저장하는 논리적인 단위

> 파일은 실제 물리 disk(HDD, SSD)에 저장되고, 용량이 존재

> 파일은 파일명과 확장자로 식별됨

① Binary 파일

> 메모장으로 열었을 때 알아볼 수 없음

> 코드 상으로 읽고 싶으면 관련 모듈을 사용해야 함

② text 파일

> 메모장으로 열었을 때 우리가 알아볼 수 있는 파일

3) 절대 경로와 상대 경로

- 절대 경로 : 드라이브 문자를 포함한 전체 경로
- 상대 경로 : ../../../ (..은 바로 전 경로)
- 경로는 문자열이므로 반드시 \\ 두 번 쓸 것!

2. 파일을 다루는 기본 구조

1) 파일 객체 = open(“파일 이름”, “파일 열기 모드”)

- 파일 객체 : 변수와 비슷
- 파일 이름 : 컴퓨터에 존재하는 파일 명
- 파일 열기 모드 : 열었을 때 어떤 행위를 할 것인지 미리 결정

> r : 읽기 모드(read)

> w : 쓰기 모드(write)

> a : 추가 모드(add or append), 파일의 끝에 내용 추가

3, 파일 열기

1) 파일 읽기

- file = opne(“”, “r”)

- read() : 파일의 전체의 내용을 문자열로 반환

ex) file = open("C:\\Users\\atheu\\Desktop\\파이썬\\정리.txt", "r")

text = file.read()

file.close() -> 반드시 닫아줘야 함

2) with를 이용해 close() 생략하기

- with open("C:\\Users\\atheu\\Desktop\\파이썬\\정리.txt", "r") as file :

(파이썬 파일과 같은 폴더에 있으면 파일명만 입력)

text = file.read()

print(text)

3) 파일 내용 한 줄씩 읽기

① file.readlines()

② with open("C:\\Users\\atheu\\Desktop\\파이썬\\정리.txt", "r") as file :

while True :

text = file.readline()

if not text :

break

print(text, end="")

- 자동으로 다음 줄로 이동
- 한 번 읽거나 쓰고 나면 자동으로 그 다음 위치로 offset 이동
- 처음 파일을 열면 offset은 처음 위치 -> 한줄 로딩 -> 다음 줄 이동
- 원한다면 offset 위치를 변경하여 원하는 위치의 내용을 읽거나 쓸 수 있음

4) 통계 산출(파일의 단어 개수, 라인 수)

ex) with open("hi.txt", "r") as file :

text = file.read()

word_list = text.split() -> 공백 기준으로 문자를 나눈다.

line_list = text.split("\n") -> 개행 기준으로 문자를 나눈다.

print(word_list)

print("단어 수 :", len(word_list))

print()

print(line_list)

print("라인 수 :", len(line_list))

5) 파일 쓰기(w)

- 파일이 없으면 새로 생성, 있으면 지우고 새로 만든다.
- write()도 마찬가지로, 자동으로 쓰고난 뒤 다음 위치로 offset 이동

ex) with open("tttt.txt","w") as file :
    for i in range(11) :
        text = "{}번째 줄입니다.\n".format(i)
                                    -> 엔터키도 넣어줘야 됨
        file.write(text)

<2020. 02. 06>

1. 바이너리 파일

- 뒤에 'b'를 붙임(텍스트 파일은 't' 붙이는데 생략 가능)

ex)
```
file = open("C:\\Users\\Administrator\\Desktop\\0207\\a.jpg","rb")
date = file.read()
file. close()

file = open("C:\\Users\\Administrator\\Desktop\\0207\\a.jpg", "wb")
file.write(date)
file.close()
```

2. 예외 처리

- 개발자가 의도하지 않은 오류 발생에 대한 오류 처리
- try : -> 오류 발생 예상 지역
  기본 수행문(무조건 수행)
  except : -> 오류 발생 시 오류가 발생한 코드에서 except문으로 점프
  오류 발생 시 수행되는 수행
  finally : -> 마지막에는 무조건 수행되는 구문
  -> 정상이든 오류든 구분 없이 무언가 마무리할 코드가 있을 때 사용
  -> try, except에 종속됨

ex)
```
try :
    num1, num2 = map(int, input("두 수 입력 : ").split())
    print("나눈 결과 :",(num1//num2))

    my_list = [1, 2, 3]

    index = int(input("숫자 입력 : "))
    try :
        print("값 :", my_list[index])
    except :
        print("내부 try")

except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
except ValueError :
    print("숫자를 입력하세요.")
```

```
except IndexError :
  print("인덱스가 잘못되었습니다.")
except :
  print("뭔지 몰라도 에러")
```

3. 객체 지향 프로그램(Object Oriented Programming, OOP)

1) 특징

- 코드의 재사용성이 높음
- 코드 관리가 편함
- 프로그램의 신뢰성이 높아짐

2) 클래스

- 일종의 설계도 또는 틀(붕어빵 틀)
- 정의 : 객체를 정의해놓은 것
  용도 : 객체를 생성
- 속성(변수)를 정의하거나 기능(함수)를 정의할 수 있음
  > 함수와 마찬가지로 작성만으로는 프로그램 수행에 영향 미치지 않음
  > 객체(인스턴스)를 생성한 뒤부터 클래스에 작성된 효력 발생
- 메서드
  > 클래스 안에 정의된 함수
  > 메서드 생성 시 반드시 최소 하나의 매개 변수 필요
  > 보통 나 자신을 의미하는 self라는 이름으로 함

3) 객체

- 클래스란 설계도를 통해 만들어진 실제 사물(붕어빵)
- 정의 : 실제로 존재하는 것
  용도 : 클래스에 정의된대로 사용함
- 구성 요소 : 속성(ex. 색상, 크기 등), 기능(ex. 찍다, 걸다, 끄다 등)
- 객체는 클래스에서 정의한 다수의 속성과 기능을 가질 수 있음
  > 속성 : 변수 / 기능 : 함수

4) 인스턴스 : 사례, 경우, 실체

- 기본적으로 객체와 같은 의미
- 문장의 쓰임에 따라 구분
  > 클래스를 통해 실제로 만들어진 객체를 인스턴트라고 함

ex)

```
class Car :
        def drive_car(self, speed) :
                print("{} 주행 준비...".format(self.model))

                if self.power == False :
                print("주행불가 : 시동을 켜주세요.")
                return

                if speed > self.max_speed :
                        print("{}의 최고 속도는 {}km입니다. 속도를 줄이세요.“
                                        .format(self.model,self.max_speed))
                        speed = self.max_speed

                print("{}km로 주행합니다.".format(speed))

car1 = Car()
car2 = Car()

car1.model = "BMW"                      -> BMW 주행 준비...
car1.power = False                         주행불가 : 시동을 켜주세요.
car1.max_speed = 200                       SONATA 주행 준비...
                                           SONATA의 최고 속도는 180km입니다.
                                                        속도를 줄이세요.
car2.model = "SONATA"                      180km로 주행합니다.
car2.power = True
car2.max_speed = 180

car1.drive_car(180)
car2.drive_car(200)
```

↳ drive_car 호출 시 정의된 매개변수는 2개
↳ self에는 자동으로 호출하는 인스턴스가 대입
↳ 그 뒤 매개변수부터는 호출할 때 값을 전달해야 함
↳ 인스턴스라는 하나의 변수에 모든 속성을 담아놓음, 연관성이 생김
↳ 함수를 호출할 때 그냥 호출해도 self에 인스턴스가 대입되므로
함수의 수행문 안에서 self.~로 호출한 인스턴스 속성들 사용 가능

5) 생성자

- 인스턴스 생성 시 자동으로 호출되는 메서드(무조건)
- 인스턴스 생성과 동시에 속성을 추가/초기값 지정이 필요한 경우 사용
- 규칙 : __int__
- ex) class Car :

```
    def __init__(self, model) :
        print("생성자 호출")
        self.model = model

    car1 = Car("SONATA")          -> 생성자 호출
    print(car1.model)                 SONATA

    car2 = Car("Kona")            -> 생성자 호출
    print(car2.model)                 Kona
```